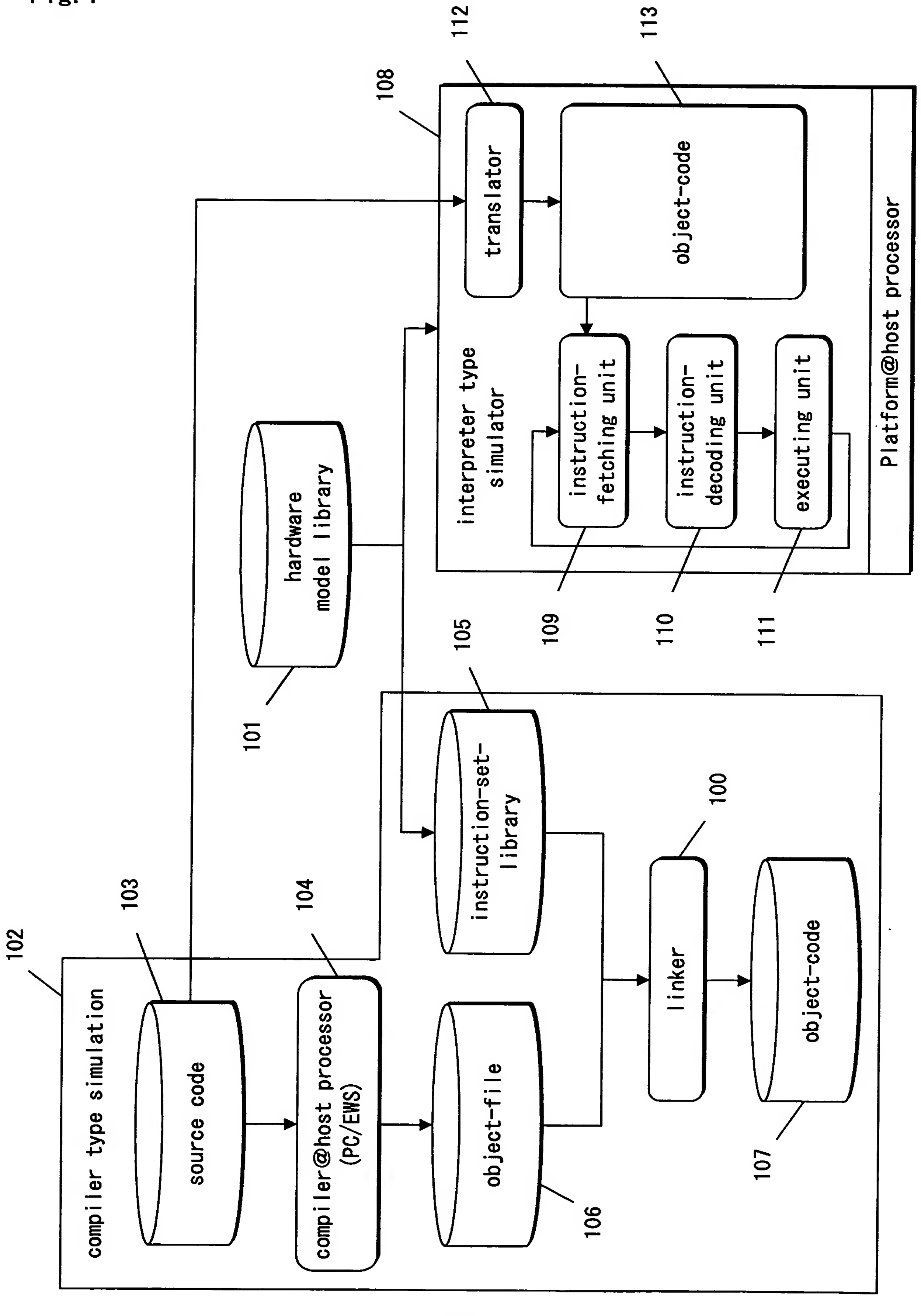
Fig. 1



102
compiler type simulation
103
source code
104
compiler@host processor (PC/EWS)
106
object-file
105
instruction-set-library
100
linker
107
object-code
101
hardware model library
108
interpreter type simulator
112
translator
113
object-code
109
instruction-fetching unit
110
instruction-decoding unit
111
executing unit
Platform@host processor

Fig. 2

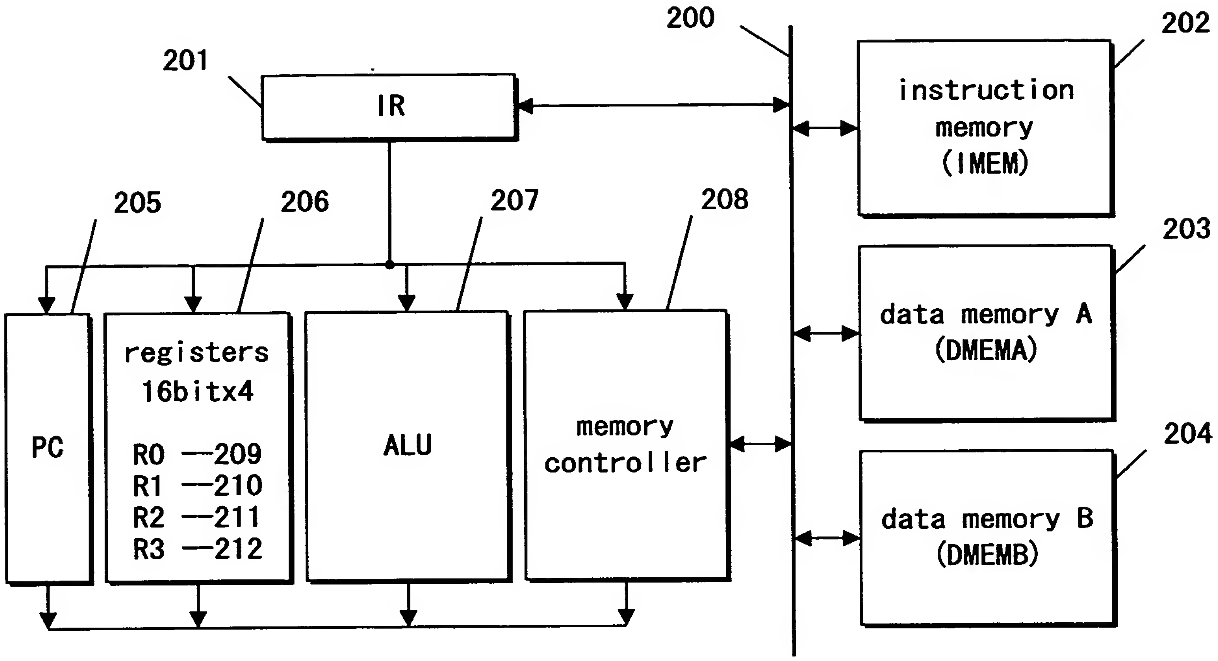


Fig. 3 (a)

instruction	first operand	second operand	machine language
ADD	Ra,	Rb	b000
SUB	Ra,	Rb	b001
AND	Ra,	Rb	b010
OR	Ra,	Rb	b011
LD	Ra,	@Rb	b100
ST	Ra,	@Rb	b101
SET	Ra,	IMD	b110
MOV	Ra,	Rb	b111

Fig. 3 (b)

register	machine language
R0	b00
R1	b01
R2	b10
R3	b11

Fig. 4(a)

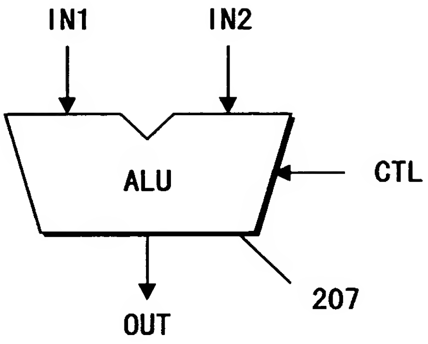


Fig. 4(b)

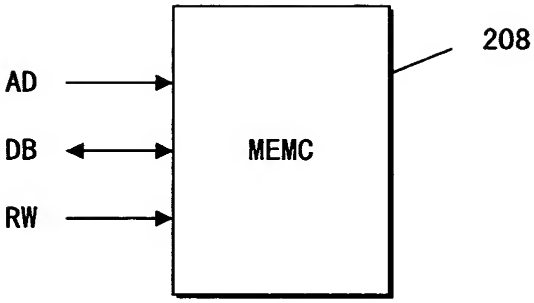
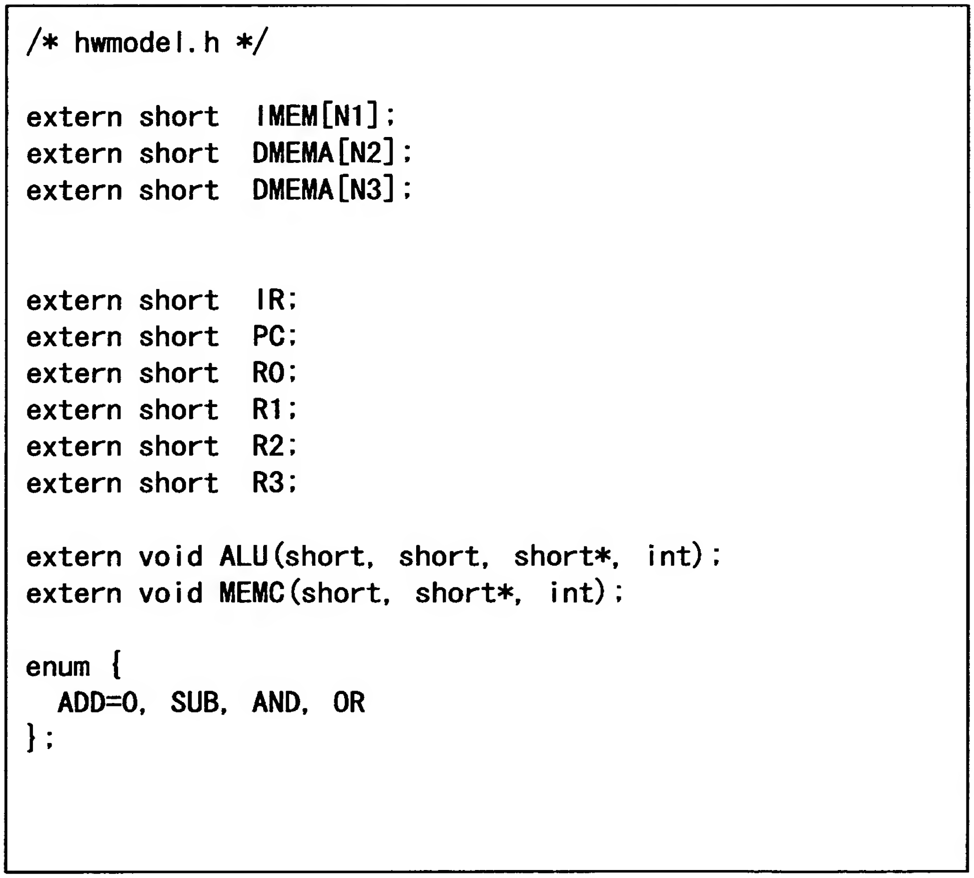


Fig. 5



```
/* hwmodel.h */

extern short  IMEM[N1];
extern short  DMEMA[N2];
extern short  DMEMA[N3];


extern short  IR;
extern short  PC;
extern short  R0;
extern short  R1;
extern short  R2;
extern short  R3;

extern void ALU(short, short, short*, int);
extern void MEMC(short, short*, int);

enum {
  ADD=0, SUB, AND, OR
};
```

Fig. 6

```
short  IMEM[N1];
short  DMEMA[N2];
short  DMEMB[N3];


short  IR, PC, R0, R1, R2, R3;


void ALU(short IN1, short IN2, short *OUT, int CTL) {
 switch (CTL) {
  case 0:     // ADD
    *OUT = IN1 + IN2;
    break;
  case 1:     // SUB
    *OUT = IN1 - IN2;
    break;
  case 2:     // AND
    *OUT = IN1 & IN2;
    break;
  case 3:     // OR
    *OUT = IN1 ¦ IN2;
    break;
  }
}


void MEMC(short AD, short *DB, int RW){
 switch (RW) {
   case 1: // Read
    if (AD < 0x100)
      *DB = DMEMA[AD & 0xFF];
    else
      *DB = DMEMB[AD & 0xFF];
    break;

   case 0: // Write
    if (AD < 0x100)
      DMEMA[AD & 0xFF] = *DB;
    else
      DMEMB[AD & 0xFF] = *DB;
    break;
  }
}
```

Fig. 7

```
#include "hwmodel.h"

void ADD(short *RS1, short RS2) {
  ALU(*RS1, RS2, RS1, ADD);
}

void SUB(short *RS1, short RS2) {
  ALU(*RS1, RS2, RS1, SUB);
}

void AND(short *RS1, short RS2) {
  ALU(*RS1, RS2, RS1, AND);
}

void OR(short *RS1, short RS2) {
  ALU(*RS1, RS2, RS1, OR);
}

void LD(short *RS1, short RS2) {
  MEMC(RS2, RS1, 1);
}

void ST(short *RS1, short RS2) {
  MEMC(RS2, RS1, 0);
}

void SET(short *RS1, short IMD) {
  *RS1 = IMD;
}

void MOV(short *RS1, short RS2) {
  *RS1 = RS2;
}
```

Fig. 8(a)

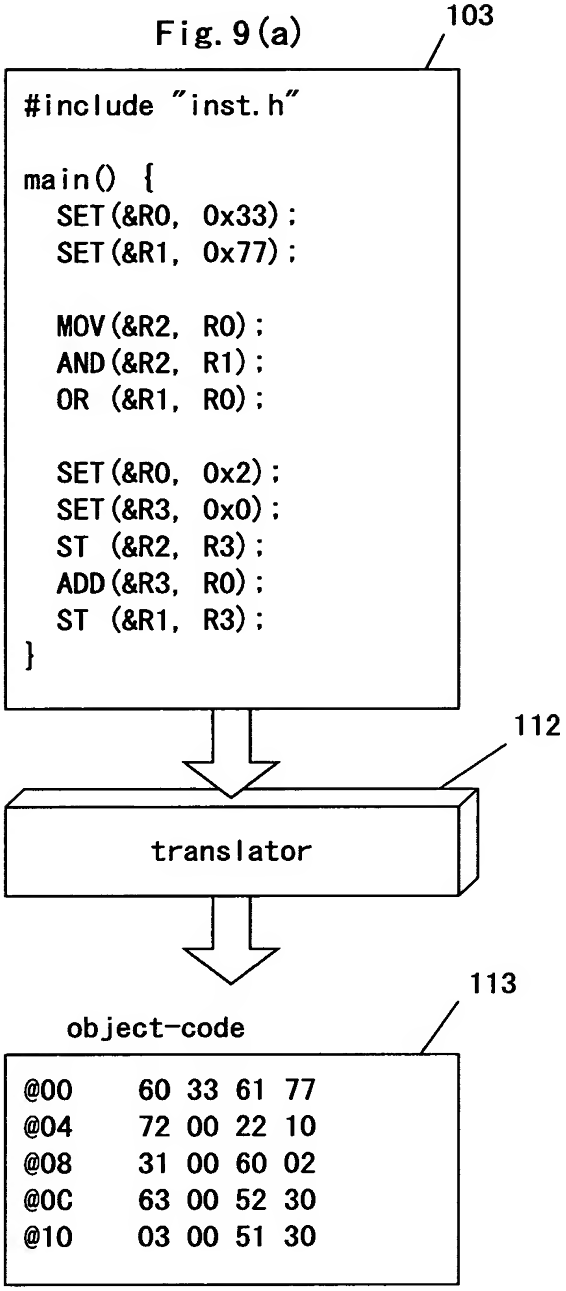
```
/* inst.h */

extern void  ADD(short *, short);
extern void  SUB(short *, short);
extern void  AND(short *, short);
extern void  OR(short *, short);
extern void  LD(short *, short);
extern void  ST(short *, short);
extern void  SET(short *, short);
extern void  MOV(short *, short);
```

Fig. 8(b)

```
#include "inst.h"

main( ) {

        . . . . .
  SET(&R0, 0x33);
  SET(&R1, 0x77);

  MOV(&R2, R0);
  AND(&R2, R1);
  OR (&R1, R0);

  SET(&R0, 0x2);
  SET(&R3, 0x0);
  ST (&R2, R3);
  ADD(&R3, R0);
  ST (&R1, R3);


}       . . . . .
```

103

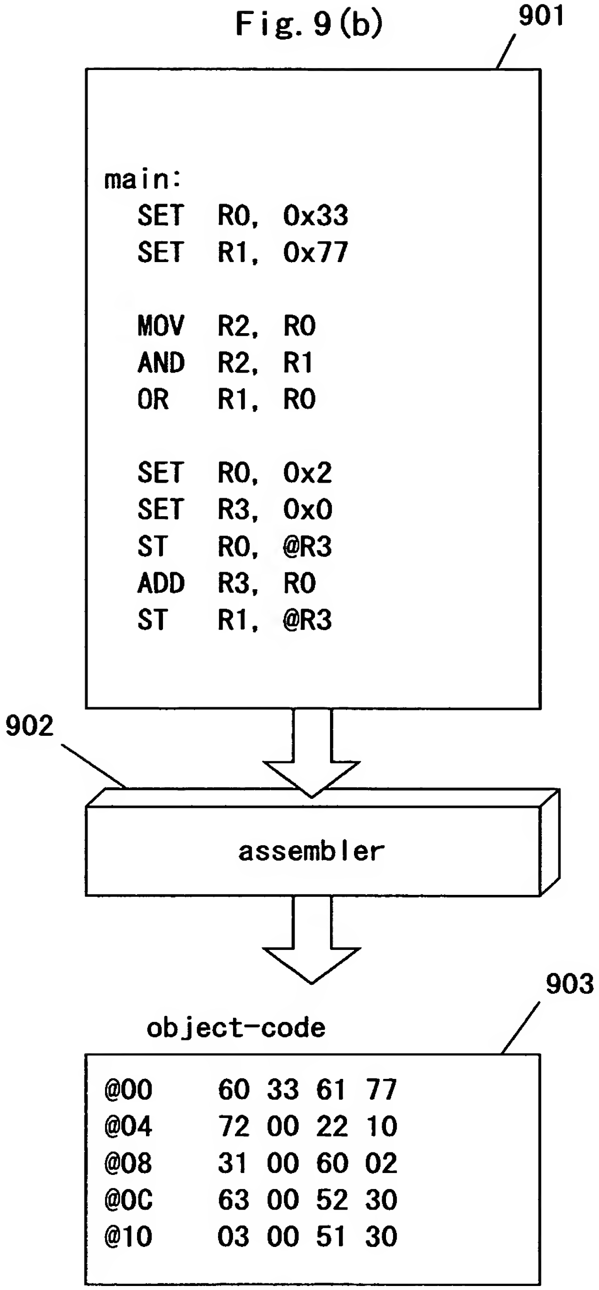
Fig. 9(a)



```
#include "inst.h"

main() {
  SET(&R0, 0x33);
  SET(&R1, 0x77);

  MOV(&R2, R0);
  AND(&R2, R1);
  OR (&R1, R0);

  SET(&R0, 0x2);
  SET(&R3, 0x0);
  ST (&R2, R3);
  ADD(&R3, R0);
  ST (&R1, R3);
}
```

```
@00    60 33 61 77
@04    72 00 22 10
@08    31 00 60 02
@0C    63 00 52 30
@10    03 00 51 30
```

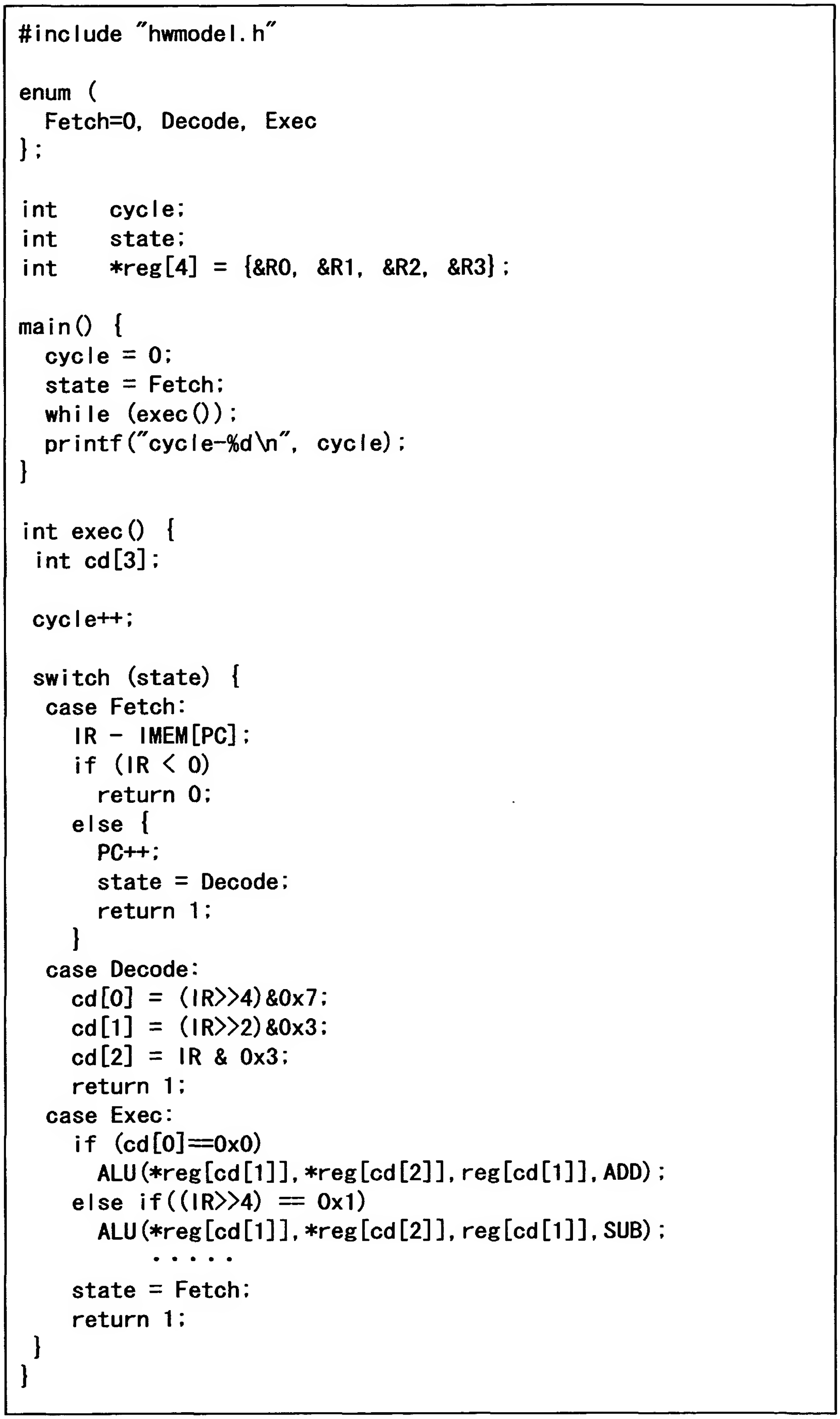
Fig. 9(b)



```
main:
  SET  R0, 0x33
  SET  R1, 0x77

  MOV  R2, R0
  AND  R2, R1
  OR   R1, R0

  SET  R0, 0x2
  SET  R3, 0x0
  ST   R0, @R3
  ADD  R3, R0
  ST   R1, @R3
```

```
@00    60 33 61 77
@04    72 00 22 10
@08    31 00 60 02
@0C    63 00 52 30
@10    03 00 51 30
```

Fig. 10



```
#include "hwmodel.h"

enum (
  Fetch=0, Decode, Exec
};

int    cycle;
int    state;
int    *reg[4] = {&R0, &R1, &R2, &R3};

main() {
  cycle = 0;
  state = Fetch;
  while (exec());
  printf("cycle-%d\n", cycle);
}

int exec() {
 int cd[3];

 cycle++;

 switch (state) {
  case Fetch:
    IR - IMEM[PC];
    if (IR < 0)
      return 0;
    else {
      PC++;
      state = Decode;
      return 1;
    }
  case Decode:
    cd[0] = (IR>>4)&0x7;
    cd[1] = (IR>>2)&0x3;
    cd[2] = IR & 0x3;
    return 1;
  case Exec:
    if (cd[0]==0x0)
      ALU(*reg[cd[1]],*reg[cd[2]],reg[cd[1]],ADD);
    else if((IR>>4) == 0x1)
      ALU(*reg[cd[1]],*reg[cd[2]],reg[cd[1]],SUB);
          .....
    state = Fetch;
    return 1;
 }
}
```

Fig. 11

```
short IMEM[N1];
short DMEMA[N2];
short DMEMB[N3];

short IR, PC, R0, R1, R2, R3;
long cycle;
double power;

void ALU(short IN1, short IN2, short *OUT, int CTL){
  switch (CTL){
  case ADD:
    *OUT = IN1 + IN2;
    break;
  case SUB:
    *OUT = IN1 - IN2;
    break;
  case AND:
    *OUT = IN1 & IN2;
    break;
  case OR:
    *OUT = IN1 ¦ IN2;
    break;
  }
  cycle += 1;
  power += 0.01;
}

void MEMC(short AD, short *DB, int RW){
  switch (RW){
   case 1: // Read
     if (AD < 0x100)
       *DB = DMEMA[AD & 0xFF];
     else
       *DB = DMEMB[AD & 0xFF];
   cycle += 1;
   power += 0.02;
   break;

   case 0: // Write
     if (AD < 0x100)
       DMEMA[AD & 0xFF] = *DB;
     else
       DMEMB[AD & 0xFF] = *DB;
   cycle += 2;
   power += 0.04;
   break;
  }
}
```

Fig. 12

```
#include "hwmodel.h"

void ADD(short *RS1, short RS2) {
  ALU(*RS1,RS2,RS1,ADD);
}

void SUB(short *RS1, short RS2) {
  ALU(*RS1,RS2,RS1,SUB);
}

void AND(short *RS1, short RS2) {
  ALU(*RS1,RS2,RS1,AND);
}

void OR(short *RS1, short RS2) {
  ALU(*RS1,RS2,RS1,OR);
}

void LD(short *RS1, short RS2) {
  MEMC(RS2,RS1,1);
}

void ST(short *RS1, short RS2) {
  MEMC(RS2,RS1,0);
}

void SET(short *RS1, short IMD) {
  *RS1 = IMD;
  cycle += 1;
  power += 0.005;
}

void MOV(short *RS1, short RS2) {
  *RS1 = RS2;
  cycle += 1;
  power += 0.005;
}
```

Fig. 13

```
#include ″inst.h″

main( ) {

      . . . . .
  SET(&R0, 0x33);
  SET(&R1, 0x77);

  MOV(&R2, R0);
  AND(&R2, R1);
  OR  (&R1, R0);

  SET(&R0, 0x2);
  SET(&R3, 0x0);
  ST  (&R2, R3);
  ADD(&R3, R0);
  ST  (&R1, R3);

      . . . . .
  printf(cycle=″%d\n″,cycle);
  printf(power=″%f\n″,power);
}
```

Fig. 14

```
#include "hwmodel.h"

long   cycle;
long   code;
double power;

void ADD(short *RS1, short RS2){
  ALU(*RS1, RS2, RS1, ADD);
  cycle += cycle_tbl[0];
  power += power_tbl[0];
  code += 2;
}

void SUB(short *RS1, short RS2){
  ALU(*RS1, RS2, RS1, SUB);
  cycle += cycle_tbl[1];
  power += power_tbl[1];
  code += 2;
}

     .....

void LD(short *RS1, short RS2){
  MEMC(RS2, RS1, 1);
  cycle += cycle_tbl[4];
  power += power_tbl[4];
  code += 2;
}

void ST(short *RS1, short RS2){
  MEMC(RS2, RS1, 0);
  cycle += cycle_tbl[5];
  power += power_tbl[5];
  code += 2;
}

     .....

void MOV(short *RS1, short RS2){
  *RS1 = RS2;
  cycle += cycle_tbl[7];
  power += power_tbl[7];
  code += 2;
}
```

Fig. 15(a)

instruction	index	cycle_tbl[ ]	power_tbl[ ]
ADD	0	1	0.01
SUB	1	1	0.01
AND	2	1	0.01
OR	3	1	0.01
LD	4	2	0.02
ST	5	3	0.03
SET	6	1	0.005
MOV	7	1	0.005

Fig. 15(b)

```
long cycle_tbl[8];
long power_tbl[8];

init(){
     .....
  fp = fopen(″table″,r);

  for (i=0; i<8; i++)
   fscanf(fp, ″%d, %f″, &cycle_tbl[i],&power_tbl[i]);
     .....
}
```